Data Structures and Algorithm Analysis

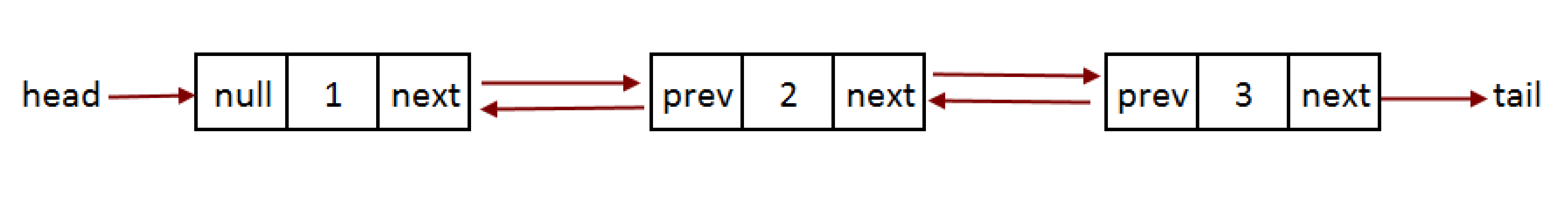
5

Dr. Syed Asim Jalal
Department of Computer Science
University of Peshawar

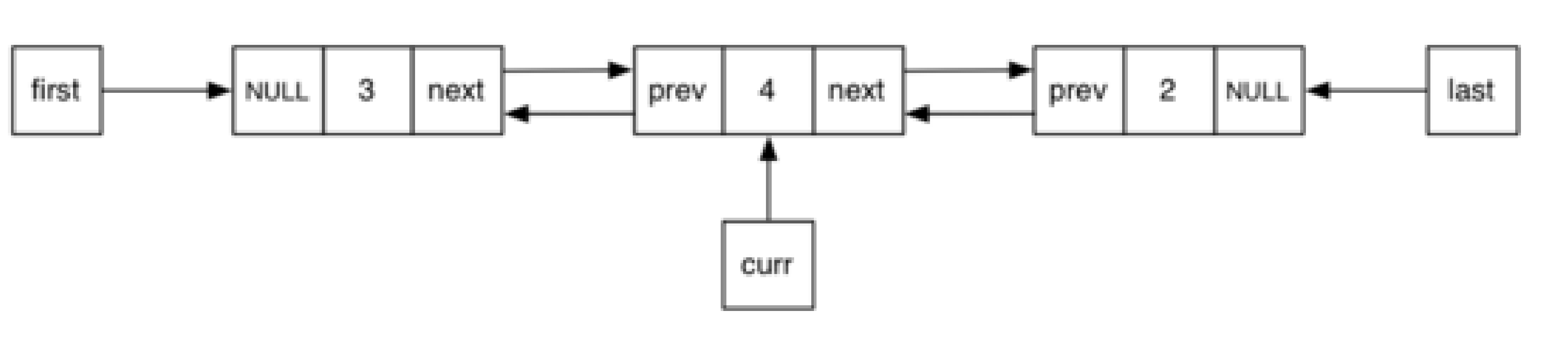
Doubly Linked List

Doubly Linked List

- Doubly Linked List is a type of a Linked List.
- Doubly Linked List has pointers to both next node and previous node.
- Each node has therefore two pointers along with information parts.
- In Doubly Linked List we can move in a linked list in both directions.



- In the pointer to previous node of the first node contains NULL and pointer to the Next node of a last node contains NULL



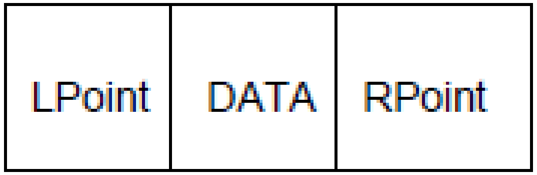
- Why we need Doubly Linked List?

1. Moving forward in a singly-linked list is easy; but moving backwards is not.
   - If we have a situation in which moving in both direction is needed, then singly-linked list is not appropriate.
   - To avoid this we can use two pointers in a node: one to point to next node and another to point to the previous node:

2. Recall that the deletion of an element at the Rear or End is not easy because we have to find the node before the tail (the last node) by link hopping.

- These two pointers help in accessing both the successor (next) and predecessor (previous) node for any node within the list.

- Every node in a doubly linked list has at least three fields:
    1. LeftPointer (or previous)
        - Pointer to the previous node
    2. RightPointer (or next)
        - Pointer to the next node
    3. DATA.



A typical doubly linked list node

Structure of doubly-linked list

```
struct dblNode
{
  dblNode * prev;
  int info;
  dblNode * next;
};

Struct dblNode * start;
```

Node creation in doubly-linked list is similar to Singly-Linked List

Creating linked list: Adding first node

- Suppose Head points to the start of a double linked list.

Algorithm:

1. Create a new node and save its address in Current
   Current ← create NewNode
2. Store information. Current → info = data
3. Current → next = null
4. Current → previous = null
5. Point Head to this first node. Set Head = Current

Adding a node at Front of a doubly linked list

To It Yourself.

- Write an algorithm that will add a new node in the start of a doubly linked list.
- You need to get help from “adding a node at Front of Singly Linked List algorithm”.
- What are the main tasks or checks to do??

Adding new nodes in the end

- For this algorithm there must be at least one node in the doubly linked list.
- Suppose **End pointer** points to the last node.

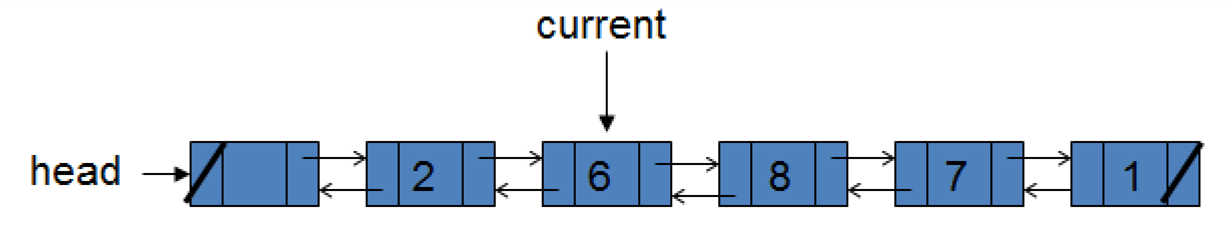
1. Create new node and store its address in **nNode**
2. Store information. **nNode**→ info = data
3. Set nNode →previous = End
4. Set nNode →next = null
5. Set **End** →next = **nNode**
6. Advance **End** to the new node. **End = nNode**

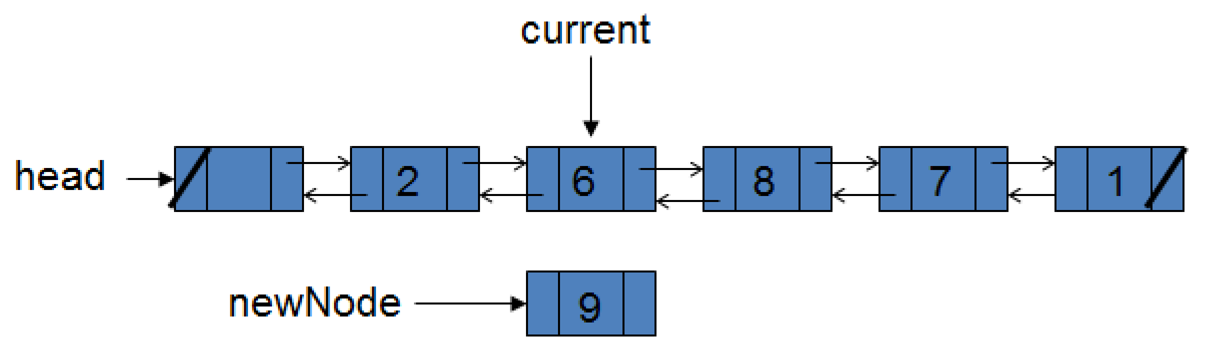
- Write an algorithm for adding node in the end of a doubly linked list in situation where we do not have any pointer pointing to the last node.
  - How would you add a node in the end?
  - What steps are needed?
  - You need to assign a Temp pointer to the start
  - Then traverse the pointer till the end till temp-> next is equal to the null.

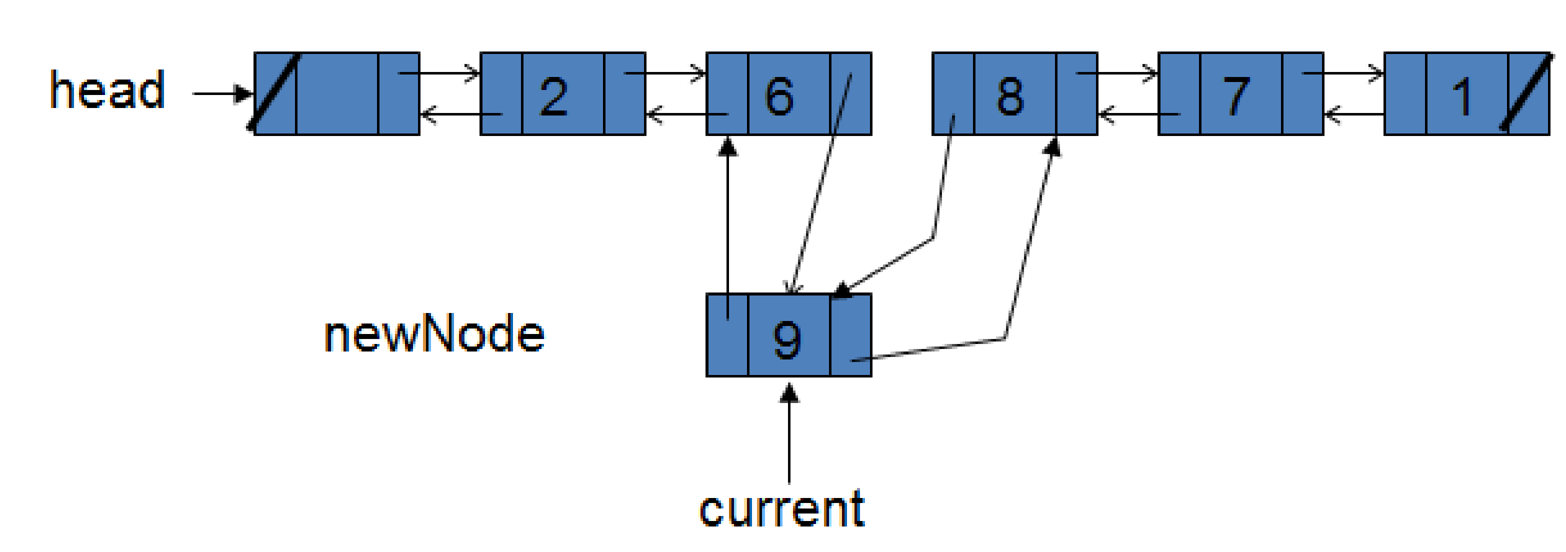
Adding a node next to a specific node

- Suppose Current points to a node and you want to add a new node to its right

- That is, adding node in the middle of a linked list.






head
2
6
8
7
1
newNode
9
current

- Suppose **Current** points to the current node, to the right of which a new node will be added. For this algorithm there must be at least one node in the doubly linked list.

1. Create new node and store its address in **nNode**
2. Store information. **nNode**→ info = data
3. Set **nNode** →previous to **Current**
4. Set **nNode** → next to **Current**→next
5. Set **Temp** to **Current**→next
6. Set **Temp**→previous to **nNode**
7. Set **Current** → next to **nNode**

The order in which pointers are reorganized or assigned is important.

Delete First Node

- Suppose Start points to the first node

1. Set Current to the Start. Current ← Start
2. Advance Start pointer. Start = Start → next
3. Delete Current. Free(Current)
4. Start→previous = Null

Delete a middle node in a double linked list

- *Delete a middle node pointed by Current*
- *p and q are temporary pointers.*

1. Save and process information in Current node
2. q = Current →prev
3. p = Current →next
4. q →next = p
5. p →prev = q
6. Delete Current

Test this algorithm on paper.

Delete node in the end of a doubly linked list

- Suppose Last points to the last node to be deleted. Temp is a temporary pointer.

1. Set Temp to last node: Temp= Last
2. Set Last to the second last node:

   Last = Temp→previous
3. Update the Next of the new last node to null

   Last→next = Null
4. Free(Temp)

Test this algorithm on paper.

- How to make one function to add a node at the beginning, middle or at the end of the doubly linked list?
  - How will you determine if a node is the first node or not?
  - How will you determine if the a node is being added to the end?

Algorithm for Forward Traversal

```
1. Set Current = First
2. Repeat Step 3 and 4 While Current != Null
3.     Process Information, print Current→data
4.     Advance Current, Current = Current→Next
5. Finish
```

Algorithm for Backward Traversal

```
1. Set Current = Last
2. Repeat Step 3 and 4, While Current != Null
3.     Process Information, print Current→data
4.     Move back Current,
                   Current = Current→Previous
5. Finish
```